CHAPTER 23

Tuning and Performance Issues

In This Chapter

How to use the Visual Studio utilities to increase application performance

Understanding Application Performance Explorer

Understanding the Stress Utility

Methods for improving performance through coding techniques

Using and understanding loading balancing

Strategies for dealing with COM+ disconnection scenarios

Most of this book has talked about putting an application together. We've also spent time addressing some debugging and packaging issues. One problem with many applications today is that the programmer stops working with the code as soon as it works. However, simply writing an application that works really isn't enough; the user also deserves an efficient and fast application.

Obviously, one problem that programmers face is getting enough time to properly optimize the application. In some cases, management is more interested in getting the product out the door, especially if there are marketing issues. So, it's important to know what to look for in an application so that you can optimize it quickly.

This chapter looks at some techniques that you can use to reduce the time and effort required to optimize an application. These techniques may not find every potential problem with your application, but they'll find the worst efficiency robbers.

The first thing we look at is the Application Performance Explorer (APE). This utility allows you to perform various types of load-testing on an application. One of the reasons this tool is so good for performance testing is that you can simulate most client, server, and database settings using the Application Performance Explorer configuration settings, rather than having to write long scripts.

APE is a tool provided with the Enterprise Edition of Visual Studio. This chapter looks specifically at the tools provided with Visual Studio Version 6. If you don't have the Enterprise Edition of Visual Studio Version 6, then skip this section and move on to the other techniques demonstrated in the chapter.

The Stress utility provides another way to check the performance of your code. Unlike the Application Performance Explorer, the Stress utility stresses the application environment rather than the application itself. For example, you can use the Stress utility to simulate a worst-case scenario platform for your application to run on, then test how your application reacts. Not only will this utility test application performance, but you can also use it to test for various types of application failures that occur when resources are low. For example, you can use it to test your memory allocation routines and determine what happens when there isn't enough memory to create the next object that your application requires. In sum, the Stress utility is one of the few dual-purpose tools that you get with Visual Studio that does both jobs well.

The next section of the chapter looks at various types of choke points that you can find within most applications. We talk about the various kinds of choke points and some of the ways you can get rid of them. The most important piece of information you can get from this section is that choke points are more often a matter of perspective, than an actual code problem. Obviously, you need to look for different areas to reduce code complexity when optimizing for code size than actual speed.

Inline code is a technique that has been used by more than a few programmers to make their applications run faster. The next section of the chapter looks at how you can implement this technique within Visual C++. We also look at when you should and shouldn't use inline code. There are some situations when using inline code might actually cost you some speed.

The final section of the chapter looks at the latest addition to the programmer's arsenal, COM+. There isn't anything magic about the code used for COM+. However, the environment it provides can create special challenges that you need to consider when tuning an application. For example, you need a way to deal with downed servers or missing connections in an efficient manner. Theoretically, a COM+ application should be designed in such a way that the user won't really notice whether there's a connection to work with or not. (We also discuss those times when theory doesn't quite match up to what you experience in the real world.)

Using Application Performance Explorer

Designing distributed applications takes a lot more in the way of planning than working with a single desktop or with a client-server architecture. Not only are there concerns about passing data across network boundaries, but now parts of your application will also execute on other machines. The problem with a distributed application is that you can't really determine in advance how such an application will perform, at least not by using the methodologies that most people use right now. The only way to really get statistics on how an application will run in such an environment is to build a test case and see how it performs. You can then extend the information you get from this test case to the application as a whole. Application Performance Explorer is designed to help you do just

that — it allows you to see how a distributed application, or at least a test case of that application, will work on your system once it's developed.

Application Performance Explorer isn't some magic forecasting tool. You need to set it up to mimic the conditions that your application will run under. Once you've set up a set of conditions, the utility runs a series of tests to see how the application will perform, given the hardware that you've provided for the utility to use. It's important to stress this last point; you need to use Application Performance Explorer on your production system unless your test system is precisely the same because the hardware configuration that you test with is going to affect the test results.

Not only does Application Performance Explorer allow you to test the design decisions you make for the application as a whole, you can also use it judge the effect of design decisions on individual components. Component technology is no longer just a nice way to promote code reuse, it's the basis of creating a distributed application. Your application can use DCOM (distributed component object model) to execute some of the components that it needs on a remote server rather than use local resources.

There are a lot of different ways to view testing. For example, you could look at individual components or you could divide the task into task areas. Application Performance Explorer uses three site-based levels of testing. The following list explains all three levels.

- **Client:** A client can be anything that uses resources or makes requests. Most of the time you'll see a client as a workstation connected to a network. However, clients could be any number of object types, including tasks running on the server itself. A server- or client-based DLL can provide a callback routine that accepts feedback from the server and generates new requests. A real-time process monitoring utility would fall into this category, as would an ISAPI Filter monitoring Web access.
- **Server:** A server is anything that provides resources and processes requests. In most cases, a server is some process running on a machine dedicated to multiple user access. However, just like the client, you can define alternative forms of servers, so it's important to understand this very basic definition of a server. For example, you could have a COM server on the client machine that processes requests for shared application data. This data might be something as simple as access requests between your application and an Excel worksheet.
- **Database:** Unlike clients and servers, the definition for a database is very restricted when using Application Performance Explorer. Your choice is between SQL Server and Microsoft Access databases. The utility exercises the data that these databases contain within the parameters set by your application programs. In other words, simply adding a database won't

suddenly provide realistic figures for what you can expect from your application. You need to provide the Application Performance Explorer with a component designed to exercise the database in the way that you intend the user to use the database within the final application.

Now that you have a better idea of what Application Performance Explorer can do and how it does it, let's take a quick look at the interface. Figure 23-1 shows what Application Performance Explorer looks like when you start it. The display shows the three site-level testing areas that I just mentioned. You can configure each level separately (we discuss the mechanics of doing so in the sections that follow). Above the three site-level buttons is a Profile drop-down list box. This is where you find a list of preconfigured test jobs. Application Performance Explorer comes with a collection of preconfigured jobs as part of the package, but you can add new jobs as needed. Finally, notice the blank edit box below the three test buttons. This is where Application Performance Explorer displays the results of any tests that you perform.

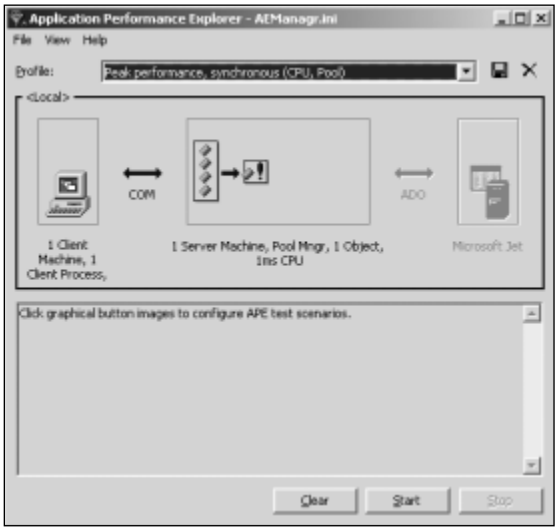


Figure 23-1: Application Performance Explorer allows you to test various configuration scenarios for distributed applications.

As you can see, the interface for this utility is relatively simple. However, configuring the utility requires a good understanding of how Application Performance Explorer performs the requested tests and how you expect your application to work. Make sure you spend some time working on the design of your application before you start testing various scenarios. Of course, you won't want to start coding until you know that your application has a good chance of working as anticipated.

The following sections tell you more about how to configure Application Performance Explorer for your specific needs.

Working with Clients

As previously stated, a client is an object that generates requests or uses resources. Of course, this is a very basic client and not very useful for testing purposes. To make the client useful, it must also be able to accept input from the server and process that input in some way to ensure the data received is the data that was expected. That's what the client configuration for Application Performance Explorer is all about.

You configure the utility to understand how requests will be generated and how those requests are to be processed when your application gets the data back from the server. Click the Client button (the first button shown in Figure 23-1) and you'll see the General tab of the Client Options dialog boxshown in Figure 23-2.

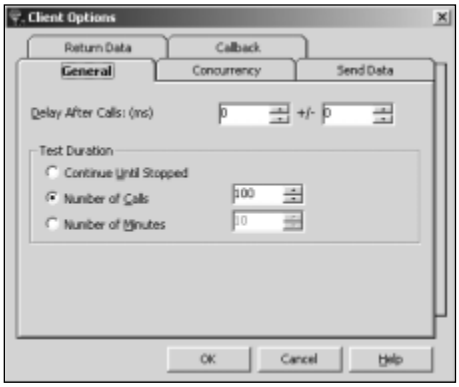


Figure 23-2: The Client Options dialog box allows you to configure Application Performance Explorer to mimic the application environment.

This is where you configure the way that Application Performance Explorer calls the server. You can tell it to keep sending requests to the server until stopped, call the server a specific number of times, or conduct the test for a specific time interval. Notice that you can also determine the interval between calls. You could set the interval short to simulate a fully loaded, high-stress situation, or extend the interval to test for nominal performance levels.

There are other ways to place a load on the application and its components. For example, you can test the application's ability to handle concurrent operations by changing the settings on the Concurrency tab of the Client Options dialog shown in Figure 23-3.

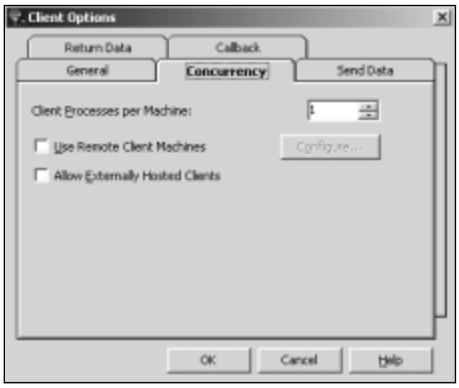


Figure 23-3: This tab allows you to place a concurrent load on the application.

The Client Processes per Machine field contains the number of processes that Application Performance Explorer will place on each client. The default client is the local machine. You can, however, add remote clients by checking the Use Remote Client Machines option. The first time you check this box, you see a Configure Remote Client Machine dialog box. This dialog box contains the names of all the remote client machines that you plan to use for the text. The Add button allows you to add more clients to the list, whereas Remove allows you to remove existing clients. The Configure Remote Client Machine dialog box allows you to create a custom test setup that takes certain types of client machine loads into account, allowing you to design your application around them.

It's important to create a client setup that matches the setup used by your business as a whole, or the target market if you're developing an application for general distribution. A developer machine may not match the machine configuration commonly used by the people who will use the application later. Make sure you create a test machine environment that reflects the actual usage conditions of your application.

There's one more checkbox on the Concurrency tab that requires some thought on your part. What if you want to see how a test works when there's more than one test being run at a time? Checking the Allow Externally Hosted Clients checkbox allows another copy of Application Performance Explorer running on another machine to perform tests on the current machine. The whole effect of running multiple tests simultaneously is to better model the business environment as it actually works. Few business users run just one application at a time; they run multiple applications. When you create a test setup that checks the anticipated performance of your custom application, you initially want to see how the application runs by itself. However, once you make this assessment, you'll also want to verify that the application will run well with other applications.

Strive for the type of application load that the user will actually encounter when using your application when running multiple tests simultaneously. Using a balanced load of CPU and disk-bound tasks may not reflect reality and will definitely degrade the feedback that you receive from Application Performance Explorer. A user survey may help you determine what applications that a user actively uses during the day. However, you may also need to resort to monitoring software to ensure that you see the actual application load on your network.

Most applications work with some type of data. As a result, it's important to test data flow from the client to the server in some meaningful way. That's where the Send Data tab of the Client Options dialog box comes into play. You'll use the options on this tab to tell Application Performance Explorer how to send data to the various server components that your application will interact with. Figure 23-4 shows what the Send Data tab looks like.

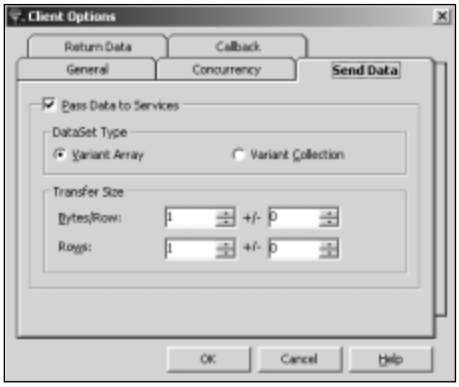


Figure 23-4: Database applications normally require some type of data flow to ensure they work properly.

The first thing you'll need to do to pass data to a service is to check the Pass Data to Services option. If you're using the asynchronous model, Application Performance Explorer will pass data to both the job manager and a worker. On the other hand, if you use the synchronous model, Application Performance Explorer will pass data to the worker alone.

In the asynchronous model, calls get made to services, but the client doesn't wait for a response. The client provides a callback function that the service uses to return a response and data from the call. On the other hand, when using the synchronous model, the client does wait for a response from the service after making a call. The response and data are normally read directly in the calling function rather than using a callback function to process the data. Asynchronous calls are normally used for streaming data like binary large objects (BLOBs), whereas synchronous calls are more appropriate for small data transfers like integers and strings.

There are two kinds of data that you can send to the service: a variant array and a variant collection. You choose the kind of data using the DataSet Type options. Microsoft recommends using a variant array because it's more efficient. All you send to the server is an array of pointers to the data — which reduces the network traffic that you might otherwise encounter passing entire collections of data. The variant collection option is provided more for testing the worst-case data transfer scenario than anything else.

Once you decide on a DataSet type, you need to define parameters for the data itself. The final set of fields on the Send Data tab define the number of bytes per row and the number of rows of data that you'll send. These values should roughly correspond to what you expect to send to the service when using the application for real. There are also boxes for determining a variation of the data size. In most cases, you want to test your application by sending what you think will be the range of data sizes (bytes per row) that your application will produce.

Most applications expect to receive data from the server when they send a request of some type. The Return Data tab looks much like the Send Data tab that we just discussed. The difference is that you're specifying the parameters for the data that the client will receive rather than what it will send.

Applications rely heavily on callback functions to receive the data that they've requested, to be notified of events, or to handle messages generated by various sources. Part of the client configuration process is to determine how the callback function (if any) will get registered with the server. That's where the Callback tab, shown in Figure 23-5, comes into play.

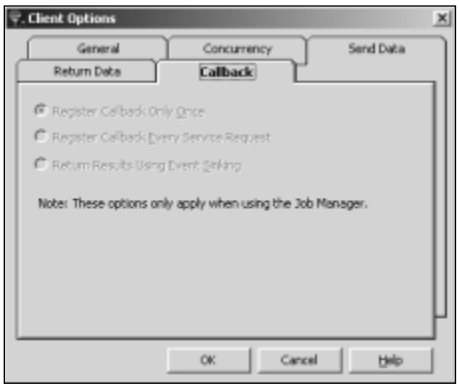


Figure 23-5: Use the Callback tab options to send data back to an application requires event notifications.

Notice the note that says that this tab applies only to applications that rely on the Job Manager. This implies that you can set the callback option only for applications that rely on the asynchronous model, which makes sense because the synchronous model relies on the application waiting for return data. There are three different ways to register your callback function. You can register the callback once, which implies that your application maintains a constant connection to the server (something

that may not happen with some types of Internet or intranet calls). Application Performance Explorer will also register the callback function with every service call, which is more appropriate for most Internet and intranet applications where the connection gets broken and the service gets unloaded when no longer needed. Finally, you can tell Application Performance Explorer that the service will fire an event, rather than perform a callback, to return the requested data to the application. This is the method used by many ActiveX components.

Working with Servers

Servers can process client requests as well as provide resources of various types. With this in mind, you need to perform two kinds of server configuration tasks when using Application Performance Explorer. The first task is to tell the client where to find the data it needs. The second task is to tell the server how to interact with the client.

Let's begin by looking at the connection process. Use the View | Service Connection Options command to display the Application Service Connection dialog box shown in Figure 23-6.

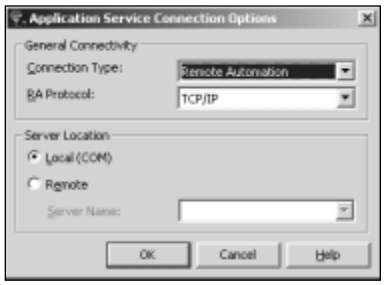
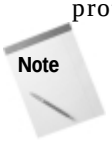


Figure 23-6: You need to create a connection with a server for many network applications.

The first thing you need to decide is what kind of connection you want to create between the client and the server. This initial connection defines the method used to launch the server and any associated components. Application Performance Explorer defaults to using remote automation because that type of connection is more commonly available now. However, you can also choose to use DCOM. If you choose remote automation, then you also need to choose a remote automation protocol (TCP/IP, SPX, or named pipes).



If you decide to enable remote automation during testing, make sure you run the Automation Manager utility. This utility passes remote automation requests from the client through the automation proxy to the appropriate Remote Automation Server components.

After you configure the connection, you need to define the server location. Application Performance Explorer defaults to using the local machine. This setting is handy for checking your settings, to make sure that they'll work, or for performing a local test rather than something that includes the entire network. If you do choose to use

a remote server (which is probably what you'll use for any real testing), then you need choose the Remote option and type a server name in the Server Name field. Application Performance Explorer will make sure you can actually create a connection before it makes the settings permanent.

Now that you have access to a server, you need to configure it for use with the client. Click the Server button (second button shown in Figure 23-1) and the DB Task tab of the Application Service Options dialog box, like the one shown in Figure 23-7, displays.

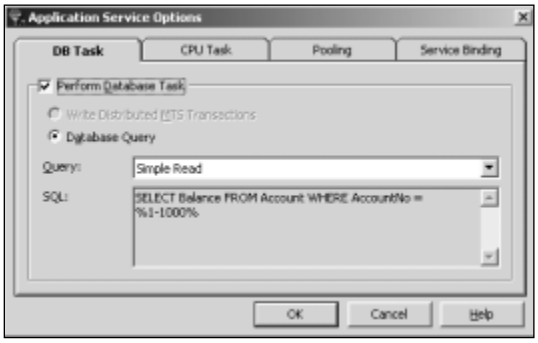


Figure 23-7: A server connection allows you to configure the client to interact with a database.

As you can see, this tab allows you to configure the server to interact with a database. Checking the Database Task checkbox tells the server to make a connection to the database (the connection is defined as part of the database configuration, which we talk about in the Working with Databases section of the chapter). There are one or two types of connections that you can make depending on the capabilities of your server. Application Performance Explorer allows you to use Microsoft Transaction Server (MTS) transactions if your server supports them. Normally, you use a predefined database query by selecting the Database Query option. Just choose the query you want to perform on the test database by selecting it from the drop-down list box.

Application Performance Explorer comes configured with a test database and predefined queries that interact with that database. If you want to create queries to interact with your database or expand the queries that you can make on the test database, then you need to modify the contents of the APETest database. The Query table of this database contains the queries that you see in the DB Task tab of the Application Service Options dialog box. In addition, you may need to modify the source code for the ACCOUNT.CLS file found in the Ape\Source\Aemtssvc folder of the Application Performance Explorer installation. This is a Visual Basic source code file that contains the queries for MTS transactions.

Besides database tasks, you can also ask the server to simulate a CPU task. Just select the CPU Task tab shown in Figure 23-8 and check the Simulate CPU Based Task checkbox.

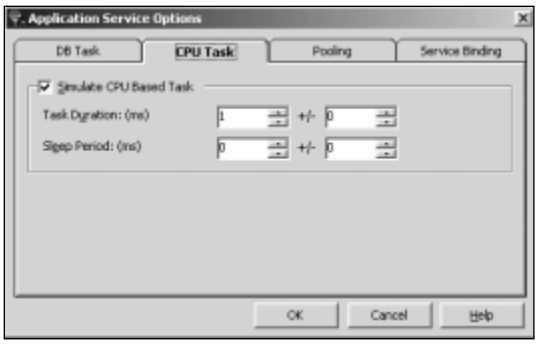


Figure 23-8: Application Performance Explorer will also allow you to simulate CPU-intensive tasks.

The Task Duration field allows you to choose the length of time that the server will simulate a CPU task. The number in the first field determines the median duration of the task in milliseconds. You can randomize the task times by adding a number to the +/– field. Likewise, you need to specify the task sleep time; in other words, how long the server will wait between CPU task simulations. As with the Task Duration field, the Sleep Period field allows you to set both a median time and the range of randomized times that the server should use. The combination of these two fields determines the relative CPU task load on the server. A high setting for task duration and a low setting for sleep time simulate a heavy load.

Resource pooling allows your server to work more efficiently. However, the application you create has to be programmed to make use of this feature. In most cases, it makes sense to use resource pooling because you want to get the most out of your server. In other cases, like a low-usage application designed for administrator use, you may not want to expend development dollars adding this feature. The kind of resource pooling also makes a difference. You need to determine whether the server will pool resources locally or use remote worker machines to get the job done. The Pooling tab of the Application Service Options dialog box, shown in Figure 23-9, allows you to make all of these resource-pooling configuration choices.

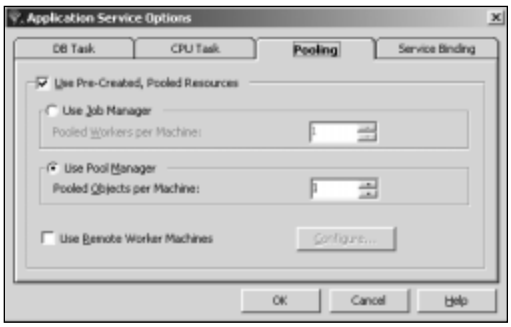


Figure 23-9: The Pooling tab allows you to simulate various forms of resource pooling.

The Pooling tab allows you to set resource pooling in motion by checking the Use Pre-Created, Pooled Resources checkbox. Application Performance Explorer lets you use either the Job Manager or the Pool Manager. If you select the Use Job Manager option, both jobs and workers are queued. On the other hand, the Pool Manager queues only objects. In other words, these options allow you to optimize the level of local or remote resource pooling used for the test. The number of objects or workers that you pool determines the stress level on server resources, with 1 being the lowest stress level.

You can also use remote worker machines when conducting tests. Click the Use Remote Worker Machines checkbox and you'll see a Configure Remote Worker Machine dialog box that allows you to add new workers to the list. As with client machines, you'll want to make sure that these remote workers actually reflect your final network setup so that the test results aren't skewed. The Add to Test button allows you to add new workers to the list, whereas Remove allows you to remove them.

The final tab in our Application Service Options dialog box is Service Binding. This tab lets you change the method used for binding objects that are used for testing purposes. Figure 23-10 shows what this tab looks like.

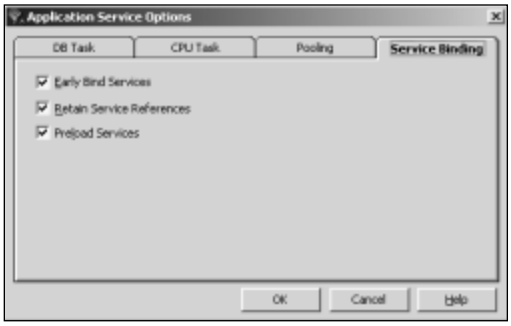


Figure 23-10: The Service Binding tab allows you to change the method used for binding objects during the test.

The Early Bind Services checkbox determines when services get bound during testing. An early bound service will improve application performance, but could limit flexibility because you can't change service properties during runtime. Checking the Retain Service Options checkbox means that workers will retain any service options required for test objects. This means that the work will have to instantiate the service only one time. After this first instantiation, the worker will use a reference to the service, which will improve application performance. Uncheck this box if you want to simulate Internet or intranet access. The Preload Services checkbox tells Application Performance Explorer to preload the server before it begins timing the test. If you'd normally preload your services, then check this option to get accurate test results. Otherwise, checking this box makes the application appear more efficient than it really is.

Working with Databases

Many applications require access to a database. However, the method and type of access required depends a great deal on how your application is set up, what environment you intend to use it in, and whether you have any legacy code to support. Application Performance Explorer provides two methods for configuring your database connection: the connection itself and how the server and client interact with the database.

Let's begin by looking at what you need to do to define a connection. Use the View | Database Connection Options command to display the Database Connection Options dialog box shown in Figure 23-11.

Figure 23-11: Use the Database Connection Options dialog box to determine the method used to create a connection to your database.

The only feature that you can change in this dialog box is the method used to create the connection. By default, Application Performance Explorer chooses the fastest and newest connection method, ADO (ActiveX Data Objects). The Access Method drop-down list box also provides you with access to the RDO (Remote Data Objects), DAO (Data Access Objects), and ODBC (Open Database Connectivity) API connection methods.

Now that we've determined what connection method to use, it's time to determine what database engine we'll use. Remember that Application Performance Explorer limits you to using either Microsoft Jet or Microsoft SQL Server. However, because the source code for this product is provided in the Ape\Source folder, you could add other database types to the list. Click the Database button (third button shown in Figure 23-1) and you'll see a Database Options dialog box like the one shown in Figure 23-12.

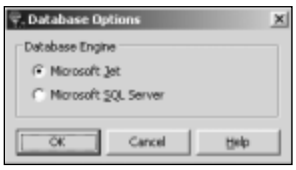


Figure 23-12: The Database Options dialog box allows you to choose a database engine to use for testing purposes.

You may look at this dialog box and feel that it's lacking a few things, such as a place to type the name of the database to use for testing. If you choose the Microsoft Jet option, then Application Performance Explorer will look for a database named

APETEST.MDB in the Ape folder. The program ships with a default test database t hat you can modify to meet your needs. It allows you to do everything from simple query changes to complex database structure changes to make the test database look more like the one used by your finished application. However, in some cases, the test database won't serve your purposes or provide very realistic test results. In that case, you'd probably need to modify the Application Performance Explorer source code.

The Microsoft SQL Server option is a little more flexible than the Microsoft Jet option. If you choose this option and click OK, you'll see an APE Database Setup Options dialog box like the one shown in Figure 23-13.

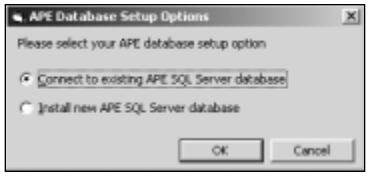

APE Database Setup Options
Please select your APE database setup option
Connect to existing APE SQL Server database
Install new APE SQL Server database
OK
Cancel

Figure 23-13: Microsoft SQL Server provides more flexibility in testing enterprise-level database applications.

This is where you choose between using the default test database that Microsoft provides or using one of your own. If you choose to use the default Microsoft database, then Application Performance Explorer will display an APE Database Connect Information dialog box like the one shown in Figure 23-14.

Figure 23-14: You need to provide a login name and password for the database file that you want to use for testing.

All you need to do is enter the name of your server, a username, and a password, and you're ready to test. On the other hand, if you choose to install a new database, then you'll see a Wizard dialog box. The following procedure shows how to install a new Application Performance Explorer database on your SQL Server.

1. Choose the Install new APE SQL Server database option on the APE Database Setup Options dialog box, and then click OK. You'll see the APE Database Setup Wizard dialog box shown in Figure 23-15.

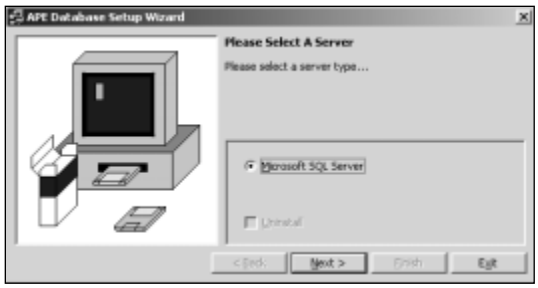


Figure 23-15: Application Performance Explorer provides a Wizard that you can use when installing a new database.

2. Click Next. You'll see the System Administrator Login dialog box shown in Figure 23-16.

Figure 23-16: The first thing to do when installing a new database is log in as the system administrator.

3. Type your server name, the administrator name for SQL Server (not for the server itself), and the SQL Server administrator password.

4. Click Next. You'll see a Destination Directory dialog box like the one shown in Figure 23-17.

Figure 23-17: You need to choose a destination directory for the installed database.

5. Ensure the default SQL Server data directory supplied in the Server Device Path field is correct.
6. Type a new Device ID value if necessary (the default value is normally correct).
7. Type a Database Size value if necessary. In most cases, you'll want to make the size greater than the default size shown. Otherwise, you may get error messages when the test database runs out of space.
8. Click Next. You'll see the Who's the DBO? dialog box shown in Figure 23-18. The database owner (DBO) is the one who actually owns the test database. This is the name to type when setting up for the test.

If you're planning on eventually making this test database into the permanent application database, make sure you use a DBO that reflects the final owner of the database rather than the test name supplied.

Figure 23-18: The choice of DBO is important if you plan to use the database after the test.

9. Type the name of the DBO in the Log on dbo ID field.
10. Type the DBO password in both the Password for log on and Confirm password fields.
11. Click Finish. Application Performance Explorer will create the new test database on the SQL Server for you. If everything goes well, you eventually see a Successful Installation dialog box.
12. Click Exit, and then OK at the Database Options dialog box. Application Performance Explorer is configured to use the new SQL Server database you've set up.

Working with Multiple Platforms

There are times when you need to develop a single application that works well on multiple platforms. It might not be economically feasible to develop separate applications for each platform that you need to support for a variety of reasons, including supported base size.

In some cases, multiple platform support means that you need to develop for the lowest common denominator. For example, you won't be able to include Windows 2000 security features in an application destined for Windows 95 use because the Windows 95 platform doesn't support many Windows 2000 security features. Because Windows 9x doesn't support advanced security, it doesn't pay to slow your application down by including that support. (What normally happens when you use a security feature that Windows 9x doesn't support is that the function call simply returns without doing anything.)

Tuning and testing are also areas where you may experience trouble getting a single application to work on multiple platforms. Windows 2000 provides better support for 32-bit applications than Windows 95/98 does; yet Windows 95/98 is better with compatibility needs. In other words, you're working with two operating systems that are tuned in different ways. Make sure you tune the application for the environment that most users will have on their workstation. In some cases, you may want to place time-sensitive functions in a separate DLL, then install different versions of the DLL depending on the operating system in use.

Another problem that you'll experience is that it's currently difficult to install more than two operating systems on a single workstation using Microsoft-supported installation methods. This makes it hard to tune your application for a variety of environments without using multiple workstations. In addition, you have to boot each operating system individually, which wastes a lot of time.

There's help for this situation in the form of VMWare. This product allows you to set up a workstation with multiple versions of any operating system—even non-Windows operating systems. VMWare creates multiple virtual machines, which allows you to run more than one operating system at a time. The environment that VMWare creates allows each operating system to believe that it's the only one running on the machine. The operating system receives part of the hard disk storage space and memory available on the workstation. In addition, each operating system environment is totally protected, meaning a crash in one operating system won't affect the others. Finally, you can create an image of a selected operating system, save it to a LAN (VMWare also supports removable media such as JAZ drives), and then download it to other machines so that all the developers in a group will have the same setup to use for testing purposes.

Recovering after a crash is fairly easy with VMWare. A feature called Checkpoint allows the developer to freeze and save the operating system environment at a specific point. If a failure occurs, the operating system is simply reloaded, allowing the developer to continue testing much faster.

There are both Linux and Windows NT versions of VMWare. You can find out more about VMWare at `http://www.vmware.com`.

Using the Stress Utility

Development machines tend to be state-of-the-art, contain lots of memory, and provide plenty of speed. On the other hand, the workstation owned by the user may not have much in the way of memory, state-of-the-art hardware, or speed. Theoretically, everyone takes his or her application to a least common denominator workstation for testing. That way a developer can ensure that the application written on a development machine looks and works just as nice on a user's workstation.

The Stress utility provides a little reality to this scenario. How many developers have both a test workstation and all of the development machines they require at their disposal? Not many, is the likely answer. So, the Stress utility allows you to simulate that memory constrained stress situation that an application you create will run into when going from your development machine to the user's workstation.

Obviously, not having to move the application from the development machine to a test workstation during the development process saves a lot of time. Even if a developer has a test workstation, there are times when it's easier to test the application on the development machine for specific types of stress conditions. Just as obviously, you still need to test the application on a workstation once you complete it to ensure that it will run as intended. Although the Stress utility does a great job of simulating an environment, the fact is that the environment is still simulated. You need to test the application on one or, preferably, more test workstations that reflect real-world usage conditions to ensure the application works as intended.

You can use the Stress utility for other kinds of testing. For example, what happens when the user's machine runs out of GDI memory? Will your application degrade gracefully or crash indignantly? Stress can help you test resource-related problems that your application may run into. You can also use Stress to test error handling when a resource isn't available and other kinds of problems that it would normally be difficult to test.

The first thing you see when starting the Stress utility is a display of the current resource values for the machine you're using it on. These values include the total amount of global memory, percentages of 16-bit user and GDI memory in use, available hard drive space, the number of available file handles, and the 32-bit memory available for user, GDI, and menu usage. Figure 23-19 shows an example of the starting display.

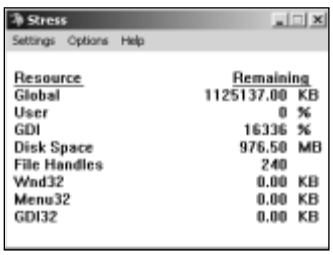


Figure 23-19: The Stress utility starting display tells you a lot about your computer.

There are two ways to test your application using the Stress utility. The first is to set fixed values for a specific resource using the Fixed Settings dialog box. You access this dialog box using the Settings | Fixed Settings command. The Fixed Settings dialog box (shown in Figure 23-20) contains a list of all the resources shown in the main window. Setting the value for a resource to –1 deallocates all Stress utility usage of that resource, whereas setting it to 0 completely allocates all of the available resources to simulate an out-of-resource condition. You can also set a field to a specific value so that Stress will allocate a specific amount of a resource. This is how you can test for workstation performance on your development machine.

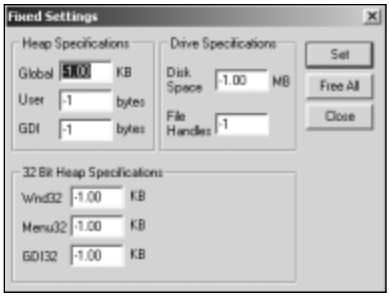


Figure 23-20: The Fixed Settings dialog box allows you to set the amount of resources that Stress will use.

The second form of testing is random resource allocation. Stress will allocate and deallocate resources in the background so that you can see how your application reacts to varying amounts of free resources. To start random resource allocation, use the Settings | Executer command. What you'll see is an Executer Options dialog box like the one shown in Figure 23-21. Like the Fixed Settings dialog box, this one contains entries for each of the resource categories displayed on the main window — only this time you see checkboxes in place of fields that you can use to set specific values. You also need to set a stress level from 1 to 4 (with 4 being the most difficult for your application to endure), the seed used for determining the random allocations (using the same seed results in the same series of allocations being used), and the interval between allocations. Stress also allows you to use a message-based allocation strategy. Every time a specific kind of message appears in the message queue, Stress will make an allocation. This setting allows you to test for event-related problems in your application.

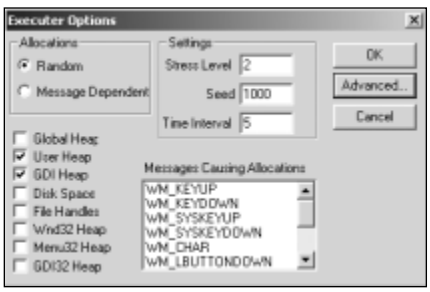


Figure 23-21: You can allow Stress to randomly set the stress level or to change it based on messages.

You can set the parameters for the various stress levels by clicking Advanced on the Executer Options dialog box. This displays a Stress Level Ranges dialog box where you can set the range of values that Stress will use for resource allocations during the application-testing process.

Finding Choke Points in Your Code

Performance has always been and will continue to be one of the goals that programmers strive for when creating an application. Of course, the level and kind of performance you actually require depends on the application you're creating. For example, a desktop application developer is less concerned about application size and more concerned with load times, memory usage, and overall application speed. On the other hand, someone developing for the Internet will definitely need to concern themselves with application size. There are many situations when optimizing for application size means giving up some application speed, so it's important to know what kind of performance you're looking for from the outset.

One book probably can't discuss every potential cause of code inefficiency, and it's unlikely that a single book could discuss all of the effects of the causes that it does cover. This section covers some of the more common problems listed below.

- Compiler Effects
- Linking Effects
- Common Coding Errors
- Threading Issues

Obviously, the question of what determines performance — the efficiency with which an application performs its task — depends a great deal on the situation. It's important that you define some criteria for testing the efficiency of your application based on the job it's designed to perform and on its environment. Defining these criteria in advance will make the job of testing the application a lot easier and reduce the risk that you'll add optimizations that don't make sense for the current situation. The following list provides an overview of the kinds of performance issues that a programmer faces.

- **Loading Time:** Consider what would happen if a medical device required five or six minutes to boot. Obviously, the patient might die before the required help became available. Although this is probably an extreme example, loading time does count in some situations.
- **Execution Speed:** Getting the job done fast is probably the most universal criteria used to judge performance.
- **Size:** Anyone who has waited more than a few minutes for an ActiveX control to download from the Internet can attest to the importance of executable file size.

- **Memory Usage:** Users generally use more than one application at a time. This means that several applications have to compete for a limited amount of memory resources. Making an application memory efficient reduces the chance that it'll end up in the swap file rather than in main memory where it can execute quickly.
- **Processing Efficiency:** Real time applications are becoming more and more prevalent as companies move to the Internet. It's no longer possible to wait 15 or 20 minutes to find out about the status of a part — people want to know now.
- **Connectivity:** Applications that make connections to both local and remote resources require a special kind of efficiency that allows them to continue working no matter what the speed of the resource is. A remote resource could exist on a high-speed network, over a dial-up connection, or may not exist at all because the user is working offline.
- **Programmer Performance:** It doesn't matter how fast and small an application is if the programmer creating it doesn't get the job done in a reasonable amount of time. Sure, a programmer could spend weeks or months fine-tuning the code for a large application, but that isn't efficient. An application development plan has to include programmer, as well as application, efficiency.

Code can have choke points that prevent it from working as anticipated. Choke points can include a wide variety of causes and effects. For example, declaring too many critical sections in your code can reduce access to it by multiple threads to a crawl. Not only do the critical sections incur processing overhead, but the machine is also prevented from processing more than one thread's requests at a time.

Common programming errors, such as initializing a variable within a loop, can create situations where your application uses too many processing cycles to perform a mundane task. These types of errors reduce the number of processing cycles available to perform application critical functions. Unfortunately, even the best programmer can make simple mistakes, which makes it imperative that you include a separate coding step that looks for coding inefficiencies. Fortunately, some of the tools provided with Visual C++ are designed to help you profile your code and to help locate areas where there may be processing problems.

Programmer efficiency is just as important as the application efficiency we've discussed in this chapter. Using the /Yc (created precompiled headers) and /Yu (use precompiled headers) switches when working with Visual C++ will improve compiler throughput. Most programmers will see anywhere from a 10 to 30 percent improvement in compiler throughput, making it more convenient to create a new build of your application when needed. Make sure that you avoid the /YX (automatic use of precompiled headers) switch because it actually reduces compiler efficiency.

Sometimes the choke points in your code have nothing to do with either your code or the environment you create for it. A compiler switch could make the difference between code that works fast and code that simply uses too many clock cycles

performing menial and unneeded processing. In addition, the definition of efficiency varies according to the current situation. An application created for Internet use should be as small as possible with a limited reliance on very few external DLLs. On the other hand, an application created for the desktop needs to load fast and work efficiently once loaded. Application size is less of a concern when the increased amount of code improves overall application efficiency.

Obviously, you need to consider what kind of efficiency you need as part of the optimization process. A strategy that works fine in one situation may lead to disaster at other times. With this in mind, you'll want to use the pointers in this section of the chapter carefully—keep the application requirements in mind. The following list provides an overview of some of the newer Visual C++ compiler switches and how they can help you improve the efficiency of your applications.

/OPT:WIN98 — This switch is turned on by default. It allows Visual C++ 6 to create applications that are more efficient when working with Windows 98 by using 4KB sections, instead of the more standard 512-byte sections that were used by previous versions. The plus side of this switch is that Windows 98 will load the application faster and may not require swap file space. The down side is that this switch increases the size of your application by a substantial amount (the actual amount varies by application). Using the /OPT:NOWIN98 switch reduces the size of your application to Visual C++ levels, or even slightly below.

Be sure to look at all of the /OPT switches that are available for both the compiler and linker. Most of these switches have been around for quite some time, but programmers simply forget that they're there. Becoming aware of all of the switches at your disposal is one way to improve both programmer productivity and application efficiency at a very small cost in time and effort.

/DELAYLOAD:<DLL Name> — Most applications rely on more than one DLL to perform service functions. Until now there were two ways to deal with these external DLLs. Either you could use the default of loading them with the application, or you could explicitly load the DLL, which means a lot of additional coding. The /DELAYLOAD switch allows you to specify that Windows not load a DLL until needed, yet without adding any loading code to slow your application down. It's still a good idea to add some exception code in case the DLL isn't present on the user's hard drive or is corrupted in some way.

/DELAY:UNLOAD	Use this switch with the /DELAYLOAD switch when you want Windows to automatically unload DLLs when the application is finished using them. This switch affects only DLLs that were loaded with /DELAYLOAD, so you don't have to worry about an important DLL getting unloaded at the worst possible time.
/INCREMENTAL:NO	Some switches require you to give up one form of efficiency for another. Incremental linking is turned on by default with Visual C++ 6 because it's such a great programmer productivity enhancement. On the other hand, incremental linking does increase both the size and loading time of the application. In addition, it decreases the execution speed of the final application in several ways. First, Visual C++ must add padding to the application to allow the linker to expand the functions you create as you develop them. In addition, when a function no longer fits in a space, the linker may add jump thunks to handle the relocation of functions to new addresses.
/WS:AGGRESSIVE	Use this switch with Windows NT 4 and above. It tells the operating system to aggressively trim the process memory used by the application. This makes the application's image in memory much smaller when it's not in use. Of course, this switch will also reduce application speed because it will have to request memory more often. You'll only want to use this switch when you want the application to have a minimal impact on system resources. In essence, you'll want to use this with most background tasks such as screen savers that don't need to interact with the user on a continuous basis.
/G6	There are a whole series of /G<*Number*> switches designed to create applications designed to run best on a specific platform. The /G6 switch allows you to create applications that run best on the Pentium Pro platform. Make sure you understand your target audience the kind of hardware they commonly use, and then use one of the /G<*Number*> series switches to make your application run faster on that hardware.

The /G series and /O series switches all optimize your code in some way. In some cases, such as the /GA switch, the code is optimized for the environment in which it will run. In other cases, the switch optimizes the code for particular hardware requirements. Still other switches optimize for a particular end result such as code size or speed. Visual C++ chooses many of these switches for you by default, but it always pays to know what Microsoft views as a requirement for a particular release type. Remember, there are no wrong choices when it comes to switches, just choices that work better in a given situation.

/SWAPRUN:{NET | CD} The /SWAPRUN switch tells the operating system to copy the executable image to the swap file first, then run it. This switch is specifically designed to optimize applications running from a CD or a network. If you specify the NET option, then the operating system will look for the file on the network. Likewise, if you specify the CD option, the operating system will load the file from the CD. This switch is usable only on Windows NT version 4.0 and above.

There's one very interesting switch that isn't new, but which is underused by many programmers. The /VERBOSE linker switch allows you to see all of the link messages during the link process. This may not seem to provide much except increased linking time because the linker is now engaged in printing a lot of messages, in addition to creating the executable image. However, all you need to do is spend several hours looking for the cause of an ambiguous link message to realize the benefit of getting complete link information. Though you wouldn't want to use the /VERBOSE switch all of the time, using it when looking for the cause of an error can save you a great deal of time.

You can use the undocumented /Y3 command-line switch to display the build times for each file in your application in the Visual C++ 6.0 Output window. Obviously, because this switch is undocumented, you can't count on finding it included with newer or older versions of the product.

Hardware Makes Tuning Difficult

Throughout this chapter, I've talked about various software problems that you'll run into when tuning your application—everything from the compiler switches that you use to the target operating system. Hardware is another potential area of concern. Consider a relatively straightforward component of the target system, such as the processor. Early test results on the Pentium III show that this processor won't do much for you unless you make use of the single instruction multiple data (SIMD) extensions that it provides. In fact, the version of the Pentium III released by Intel at the time of this writing works no better than a Pentium II running at the same clock speed unless the application includes SIMD support.

Obviously, the next processor problem that will affect programmers is the 64-bit processor being introduced by Intel. According to current Intel documentation, the Merced processor will be released in mid-2000. Intel has already made software simulators available to companies that need one, and development platforms will be available soon. At a recent demonstration, Intel was able to boot six of eight supporting operating systems for this processor, including Windows NT, Solaris, HP-UX, Irix, Novell's Modesto, and the SCO/IBM operating system code named Monterey. The bottom line is that a 64-bit processor will complicate the application development picture because this processor will require special support in your multiplatform applications.

Even if you're targeting only a single processor type, there are still going to be hardware problems to consider. Fortunately, the days when a software designer had to worry about mundane timing issues, such as when to write to video, memory are gone. On the other hand, there are other issues to consider, especially when it comes to multimedia. For example, what happens when your company has to support more than one speed of modem or graphics card? Trying to write data too quickly could result in lost information, whereas writing the data too slowly could force the hardware to use two processing cycles instead of one to process the information. Unfortunately, without the assistance of a good system engineer, you may find it difficult to solve some types of timing related hardware/software interface problems that rob your application of performance.

There are some ways to at least build a bridge between software and hardware so that timing and other tuning issues can be accounted for as part of the software design process. One of the newer methodologies is a standard being proposed by the Object Management Group called Unified Modeling Language (UML). (The technology was originally developed at Rational by Grady Booch and Jim Rumbaugh in an attempt to unify several complementary exiting modeling methodologies; Ivar Jacobson later joined them and added other modeling language elements to the mix.) In essence, this standard provides techniques for creating a single diagram that shows both the hardware and software elements of a design project, making it much easier for the software developer to tune his or her software to the target hardware. Think of UML as computer-aided software engineering (CASE) with a twist.

The best part of UML is that it specifies a method for diagramming a process, not the process itself. As a result, you could use UML as a standard way to diagram a lot of different process types—not just software- or hardware-related processes. For example, you could use it to add business logic to your diagrams, along with the more common software engineering diagrams.

UML isn't programming-language specific. This means that UML is very flexible when it comes to modeling the programming problem, but not very useful when it comes to modeling a specific solution to that programming problem. You'll also find that UML is somewhat limited (at least without some extensions) when it comes to modeling specific tools used to implement a programming problem solution. Fortunately, neither of these limitations will reduce the capability of UML to help model all of the characteristics of your next application. You can learn more about UML at `http://www.omg.org/news/pr97.htm`.

Using Inline Code

Using good programming methods and employing all of the right switches when compiling and linking are the two most common methods for improving the efficiency of your application. However, there's one optimization technique that deserves special mention: using inline code. Inline code allows you to create code that looks modular in appearance but acts as a single large conglomerate in execution.

What is inline coding all about? Normally, when you make a function call, all of the current processor state information is saved on the stack and execution begins at a new location. Saving this information takes time, as does the requirement to provide access any passed variables. In short, function calls can reduce the efficiency of your application by forcing it to perform work that has nothing to do with the problem at hand. Inline coding can reduce or even remove this inefficiency from your application.

The Enterprise and Professional Editions of Visual C++ are the only editions that support the various inline coding keywords discussed in this section. Users of other Visual C++ editions need to avoid these keywords.

Inline coding is easy to understand. Each time you make a function call within your code, Visual C++ replaces the function call with the equivalent code from your application. In other words, your application executes the code directly without the overhead of making a function call. However, this also means that you need to be very careful about the way you write your application code because direct code replacement means that your function code no longer exists on its own. Using this technique makes your application run faster, especially in situations when a small function gets called a number of times in a single application.

There are times when inline calls fail. The Visual C++ compiler will use standard function calls when it's unsure about the viability of using inline-coding techniques after it performs a cost/benefits analysis. In addition, the compiler won't inline the code if the function is too large or the function address is already taken.

Creating an inline function is relatively simple. All you need to do is include the `inline`, `__inline`, or `__forceinline` keyword as part of the function declaration. Here's an example of an `inline` keyword.

```
inline void myfunction()
{
    messagebox("This is an inline function.");
}
```

As previously mentioned, the compiler normally performs a cost/benefit analysis of your code to determine whether inline code will result in improved performance with a minimal impact on code size. However, you can override the normal analysis process using the `__forceinline` keyword. Even the `__forceinline` keyword has limitations. You can't inline code for a function even using the `__forceinline` keyword when your code is:

- ✦ Was compiled using the /Ob0 (disable inline expansion) switch, which is a standard switch for debugging. This limitation applies to the caller as well.
- ✦ Uses a different form of exception handling than the caller. The two forms of exception handling are C++ and structured.
- ✦ Has a variable number of arguments. In other words, to inline code, the number of arguments must be known during compile time and must remain constant.
- ✦ Contains inline assembly language code and you haven't compiled it using the /Og (global optimization), Ox (full optimization), O1 (minimize size), or O2 (maximize speed) switches.
- ✦ Returns an unwindable object by value and you haven't compiled it using the /GX (enable exception handling), EHs (use the synchronous exception handling model), or EHa (use the asynchronous exception handling model) switches.
- ✦ Receives a copy constructed object by value and you compiled it using the /GX, EHs, or EHa switch.
- ✦ Doesn't include the #pragma inline_recursion(on) directive. (This limitation applies only to recursive functions.)

The default call depth for #pragma inline_recursion(on) is eight. You can use the #pragma inline_depth(<*Number*>) directive to set the number of levels higher or lower.

Inline code isn't a panacea; it's simply another coding technique in your arsenal. A major problem with using inline code is that your application greatly increases in size. Every inlined function gets repeated each time that you use it. This means there could be a myriad of copies of a single function within your code.

COM+ Tuning through Load Balancing

So far, this chapter has looked at various forms of tuning from a standard programming perspective. This is the same perspective that programmers have had for quite some time, even though the tools and techniques for creating efficient programs are much better.

There's another way to tune an application that involves a major concern for the enterprise. What if the application you create won't run on a single server for everyone in the company no matter how efficient you make it? At this point, you need to worry not only about tuning the application, but tuning the way it works across multiple servers. It turns out that the capability to scale is one of the major reasons to use COM+. Your application may really have to run across multiple machines, but using COM+ allows the user to think that their version of the application is always running on the same machine.

The following sections look at some of the issues you'll face when working with multiserver applications. The first section deals with the goals that load balancing is trying to achieve. In the next section, we look at the theory behind the inner workings of load balancing. Finally, we look at an issue that many of us have had to deal with too often — downed servers. COM+ actually provides some techniques for getting around this problem.

Understanding Load Balancing Goals

Enterprise applications have to service a lot of users that may or may not be within easy access of the server. The same application that allows a salesperson to take orders over the phone has to deal with salespersons on the road and even in different countries. In fact, with the emergence of the Internet, the salesperson may not even be human — it might be the software at the other end of a buyer's query. So, the first goal of load balancing is to scale an application so that it can run on one or more servers equally well.

Salespersons on the road and users who make contact with your company over the Internet have one thing in common: They're both using a dial-in connection that could fail at any time. Recent articles in the trade press have made it clear that companies need some type of fault tolerance for online applications. Some users are actually considering not using the Internet anymore because of poor service in the past, and you don't even want to think about the loss of a large sale because the salesperson couldn't get in touch with the company database. The second goal of load balancing is to create applications that can tolerate server failures and make those failures transparent to the client (as much as possible at least).

Finally, the COM+ load-balancing strategy is designed to make it easier to work with complementary technologies. Obviously, Microsoft hasn't managed to take over the world, yet, so it's important that the enterprise-level applications you create work with the products that other companies produce.

How Does Load Balancing Work?

In the past, a client would contact a server directly to access the contents of a database or to make a new database entry. There are two problems with this approach. The first problem is that the client needs to know a specific server name and its location. Obviously, if you're running an application across multiple servers, this could present a problem to both the network administrator and the application programmer.

COM+ takes a different approach to the whole problem of access. Instead of accessing a server, the client accesses a router. The router then uses its knowledge of the available servers to connect the user to the server with the highest availability and the smallest processing load. The router receives constant feedback about

server availability through a response time tracker, which makes it easier for the router to figure out which server to use. Obviously, because every client is accessing the same router, they don't need to know the location or name of the server that they're using. Everything occurs in a dynamic environment.

As of this writing, a load-balancing setup contains one router and up to eight servers (called an application cluster). The combination of COM and MTS allows these machines to work together in such a way that the user gets the fastest possible response from the distributed application. Obviously, there are applications that require more than eight servers and perhaps more than a single router for management purposes. Microsoft will likely expand the load-balancing capabilities of Windows 2000 sometime in the future.

Don't get the idea that load balancing will affect an entire application. You might have part of the application installed on the server, another part installed on the user machine, and still another part installed on a backend server for database access. Load balancing takes place at the class level. You have to specifically mark a class as a candidate for load balancing, install it on the server. When the client calls on the component containing the class, Windows 2000 will see that the class is supposed to be load balanced and handle it appropriately.

There are some hurdles that you need to overcome when creating load-balanced COM+ classes. Because server resources are at a premium, you should use this technique for short-lived classes only. For example, a class that makes a database query, passes the results back to the user, and then destroys itself, falls into this category. In addition, the class can't assume that it will be invoked on a specific machine. This means it can't rely on resources found on a specific machine, such as files. Clients will need to observe the short-lived state of the load-balanced class as well. In other words, the client portion of your application should create an instance of the object, use it, then release it as quickly as possible.

The load-balancing router actually contains two discrete services linked together through shared memory. The first is the Load Balancing Service, and the second is the Service Control Manager (SCM).

The Load Balancing Service contains a complete list of the load-balanced components in an in-memory database. The whole purpose of this service is to maintain the load-balancing router tables for the SCM. As the network administrator or developer installs or removes components, the changes get reflected in the load-balancing router tables.

When a client makes a DCOM call to the load-balancing router, it's the SCM that actually receives the request. The SCM looks up the component in the load-balancing router table, then makes a DCOM call to a server in the application cluster to fulfill the request. The server in the application cluster creates an instance of the request object, then passes the proxy for it directly to the client. At this point, the server and the client are in direct communication — the router is no longer needed.

It's at this point when the Load Balancing Service comes back in play. It makes a request of the responding server to get response time statistics. These statistics get entered in the load-balancing router table, which allows the router to choose the correct application cluster server to respond to the next component request.

Dealing with Downed Servers and Routers

The router is the primary handler of downed servers. Part of the information in the load-balancing router table marks a server as up or down. The router knows when a server has gone down because it no longer responds to statistic requests from the Load Balancing Service portion of the router. When this occurs, the server is marked as down in the load-balancing router table. The router will also assign another server to handle the requests that the down server was handling. In this way, the user isn't even aware that the server has gone down and another server has taken its place.

A downed router is the real problem in this scenario because it represents a single point of failure. You can configure the router as a Microsoft Cluster Server (MSCS) resource. In essence, you'd create a virtual machine composed of several servers, greatly reducing the probability that router would go offline. It's unlikely that all of the servers in a cluster will go offline at the same time.

Summary

This chapter shows you how to make your applications more efficient. We looked at the utilities that allow you to measure efficiency, and the coding techniques and tool optimizations that make the resulting application more efficient. In addition, we looked at some ways to make the programming environment more efficient, which results in better programmer productivity. Obviously, efficiency is in the eye of the beholder. You need to consider the tradeoffs of using a particular enhancement within the context of the programming environment. Some additional conclusions that can be drawn from the contents of this chapter are:

- The Application Performance Explorer allows you to test an n-tier application using the machines that will actually be used to run the application in a production environment. You create a link between the various machines using the Application Performance Explorer, then send messages and data as needed to acquire response time statistics.
- Application Performance Explorer allows you to create ADO, RDO, DAO, and ODBC connections to a database. However, you can access only the database using Microsoft Jet or Microsoft SQL Server.
- Tuning an application to perform well on multiple platforms can be especially difficult because even Windows platforms vary a great deal. In some cases, you need to create a single application tuned for the lowest common denominator. In other cases, you can create better application characteristics by detecting the target platform and loading one or more DLLs specially tuned for that environment.

- The Stress utility helps you simulate the machine where an application will actually run so that you can check for potential problems with memory and disk space. It does this by consuming part of the memory that your development workstation would normally provide. Not only can you perform fixed resource tests, but the Stress utility will allow you to perform random resource-level tests as well.
- Always determine what type of performance issues you need to resolve before attempting to optimize your code. Some performance enhancements will conflict with or even cancel others. Typical performance considerations include Loading time, execution speed, size, memory usage, processing efficiency, connectivity, and programmer performance.
- Compiler switches affect both programming environment and resulting application efficiency. In addition, some switches are operating-system specific, such as the /OPT:WIN98 switch used to improve application efficiency when an application is used with Windows 98.
- Always consider the target system's hardware when tuning an application. For example, the Pentium III provides access to the SIMD extensions not found in other processors. Unless you take advantage of these extensions, the performance of a Pentium III will be on par with that of a Pentium II system of the same speed.
- One method for keeping both hardware and software design criteria in the picture is to use UML to model both. UML represents a giant leap in modeling methodology and could help make tuning an application during the design phase a lot easier in the future.
- Inline code represents one of the better ways to improve application speed once you've worked out all of the other programming problems. However, this speed comes at the price of increased application size. The amount of application size increase depends on the size of the inlined functions and the number of times those functions get called.
- Many of the tuning techniques in this chapter will work only with the Professional Edition and Enterprise Edition of Visual C++. Make sure you know the capabilities of the edition of Visual C++ that you're using before trying a new tuning technique.
- COM+ offers a new level of tuning through load balancing. Essentially, load balancing allows you to spread the application load across several servers in a way that is totally transparent to the end user.
- Load balancing not only offers improved application performance, it can also improve application availability. Load balancing provides dynamic allocation of application cluster servers, making it unlikely that the user will lose access to every server at the same time. Creating a virtual machine to use as a router reduces the chance that this single point of failure will cause problems as well.

✦ ✦ ✦